Write, Run, Test Robot Code

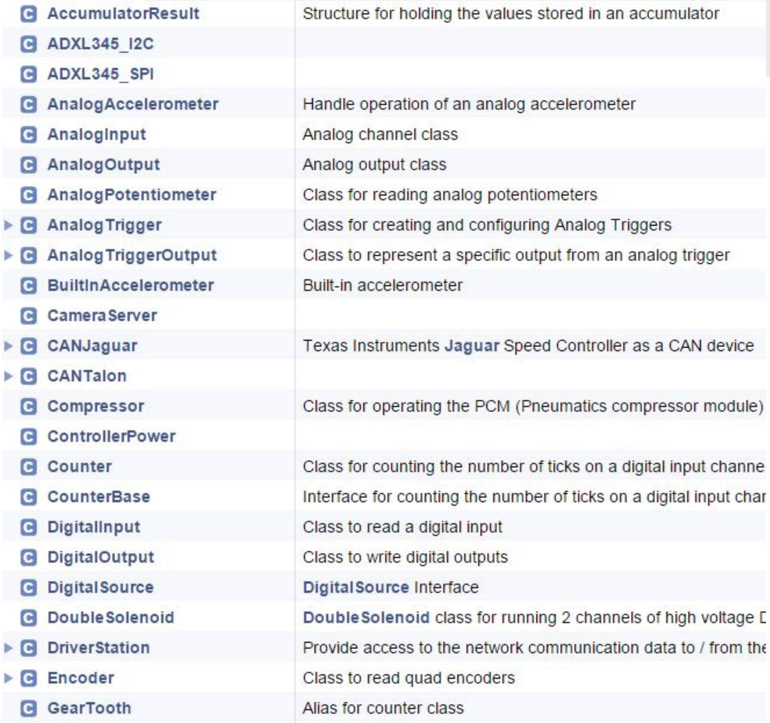
Review: Robot Parts

- **RoboRIO**: the “brain” of the robot
- **Talons**: the motors of the robot, connected to the “brain”
- **Piston**: empty canisters that move up and down by filling up with air
- **Solenoid**: the part that controls the piston’s ability to retract and extend
- **Joystick**: controls movement of robot during driver

From Robot to Code

Review: What is an API?

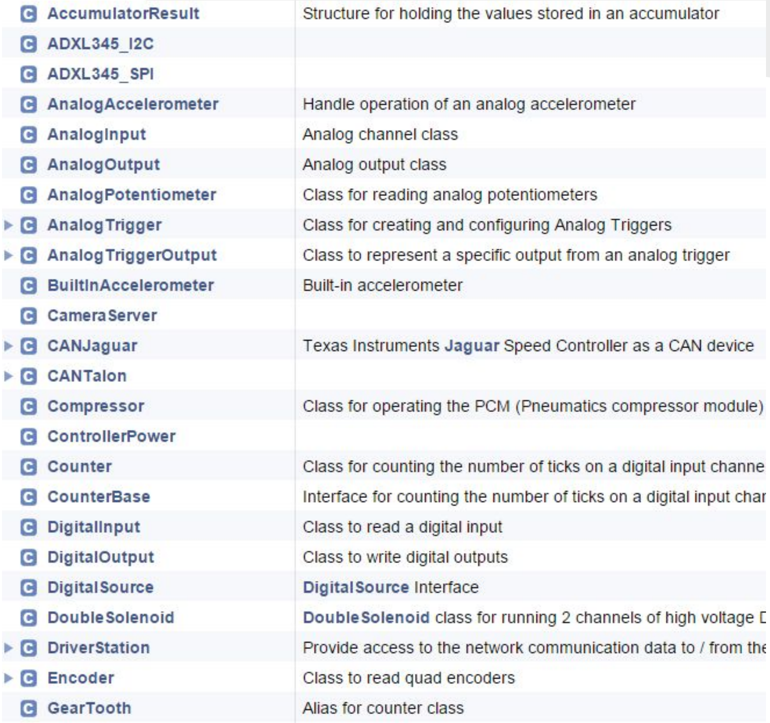
- API stands for *application programming interface*
- it provides tools to build software applications
- FIRST releases an API each season that lists robot parts and the constructors and methods to help us write our robot code.



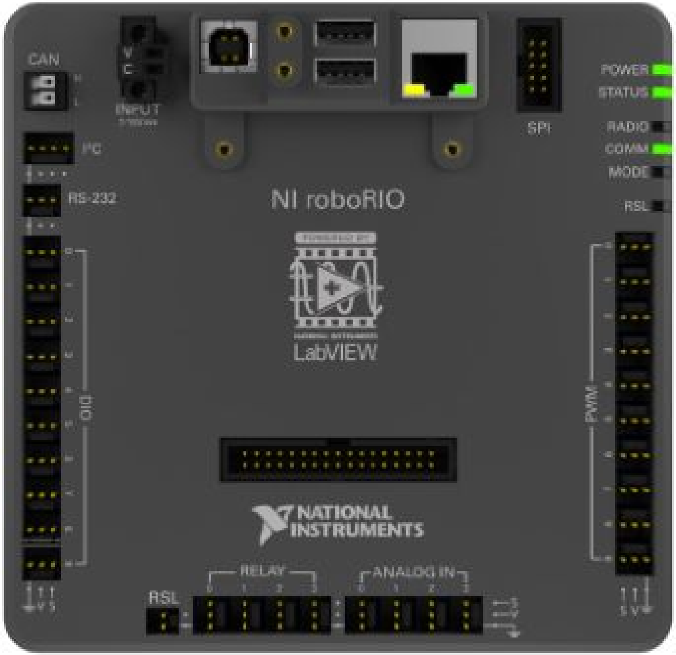
AccumulatorResult	Structure for holding the values stored in an accumulator
ADXL345_I2C	
ADXL345_SPI	
AnalogAccelerometer	Handle operation of an analog accelerometer
AnalogInput	Analog channel class
AnalogOutput	Analog output class
AnalogPotentiometer	Class for reading analog potentiometers
AnalogTrigger	Class for creating and configuring Analog Triggers
AnalogTriggerOutput	Class to represent a specific output from an analog trigger
BuiltInAccelerometer	Built-in accelerometer
CameraServer	
CANJaguar	Texas Instruments Jaguar Speed Controller as a CAN device
CANTalon	
Compressor	Class for operating the PCM (Pneumatics compressor module)
ControllerPower	
Counter	Class for counting the number of ticks on a digital input channe
CounterBase	Interface for counting the number of ticks on a digital input char
DigitalInput	Class to read a digital input
DigitalOutput	Class to write digital outputs
DigitalSource	DigitalSource Interface
DoubleSolenoid	DoubleSolenoid class for running 2 channels of high voltage D
DriverStation	Provide access to the network communication data to / from the
Encoder	Class to read quad encoders
GearTooth	Alias for counter class

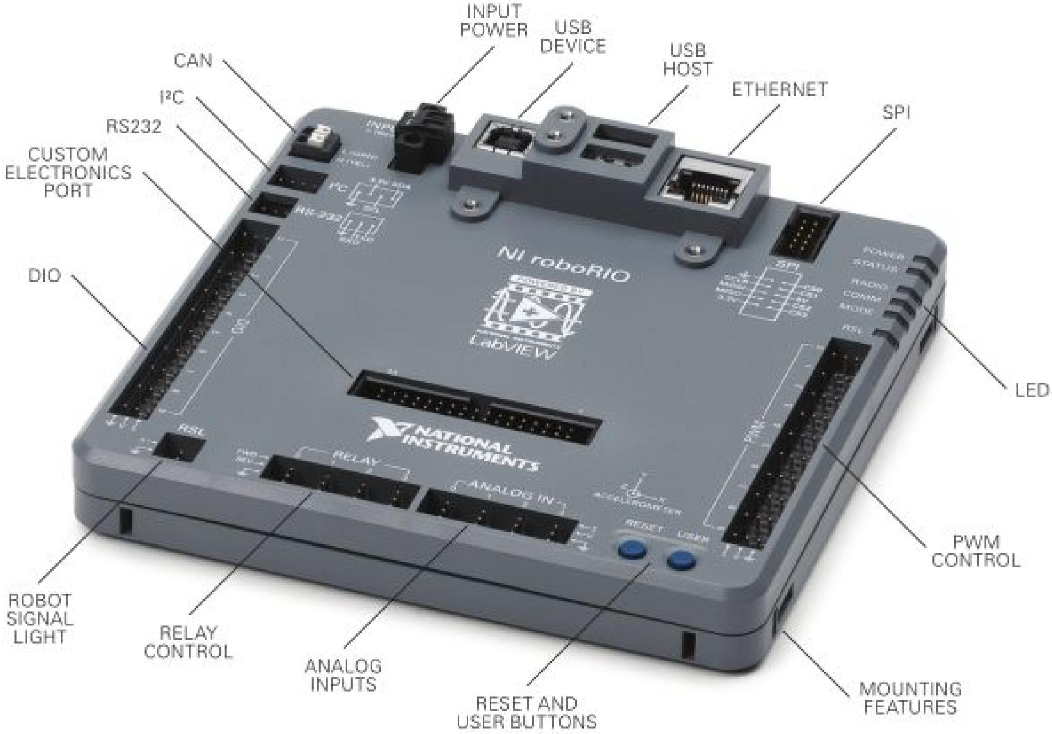
How Are Robot Parts Represented in Code?

- physical parts of the robot are represented in code by importing them and constructing them as objects
- we use methods in the API to control our objects.
- These methods come together to fulfill one big task.
- API from 2015 build season → (x)



AccumulatorResult	Structure for holding the values stored in an accumulator
ADXL345_I2C	
ADXL345_SPI	
AnalogAccelerometer	Handle operation of an analog accelerometer
AnalogInput	Analog channel class
AnalogOutput	Analog output class
AnalogPotentiometer	Class for reading analog potentiometers
AnalogTrigger	Class for creating and configuring Analog Triggers
AnalogTriggerOutput	Class to represent a specific output from an analog trigger
BuiltInAccelerometer	Built-in accelerometer
CameraServer	
CANJaguar	Texas Instruments Jaguar Speed Controller as a CAN device
CANTalon	
Compressor	Class for operating the PCM (Pneumatics compressor module)
ControllerPower	
Counter	Class for counting the number of ticks on a digital input channe
CounterBase	Interface for counting the number of ticks on a digital input char
DigitalInput	Class to read a digital input
DigitalOutput	Class to write digital outputs
DigitalSource	DigitalSource Interface
DoubleSolenoid	DoubleSolenoid class for running 2 channels of high voltage D
DriverStation	Provide access to the network communication data to / from the
Encoder	Class to read quad encoders
GearTooth	Alias for counter class





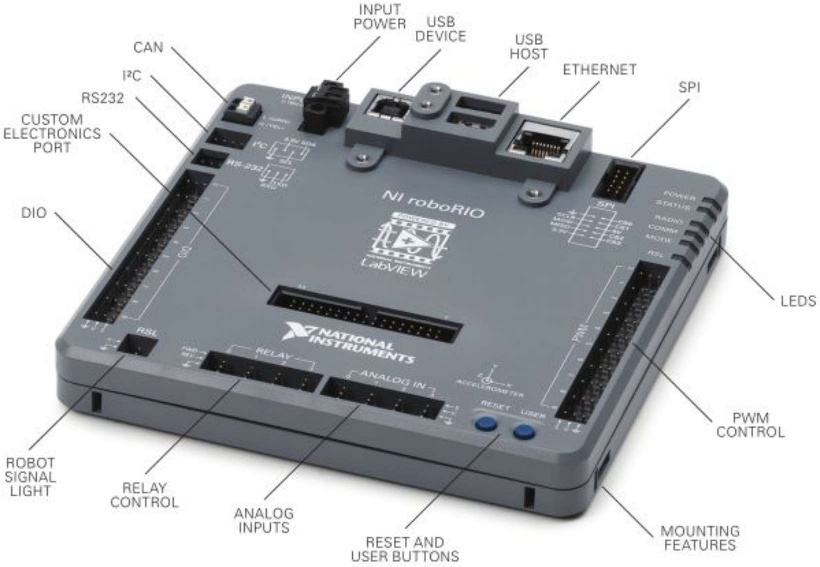
RoboRIO and its parts

RoboRIO in Code

- RoboRIO isn’t one whole object you create in robot code
- instead, each part of the RoboRIO are constructed as objects in code
- we use the parts we need in RoboRIO to accomplish our task

Find the parts of RoboRIO that are listed as objects in the 2015 API!

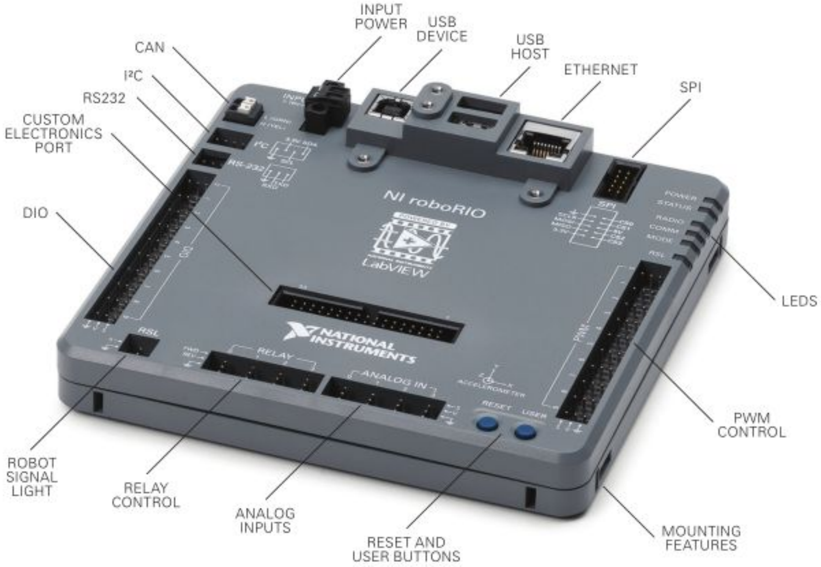
(hint: look at the picture on the right)

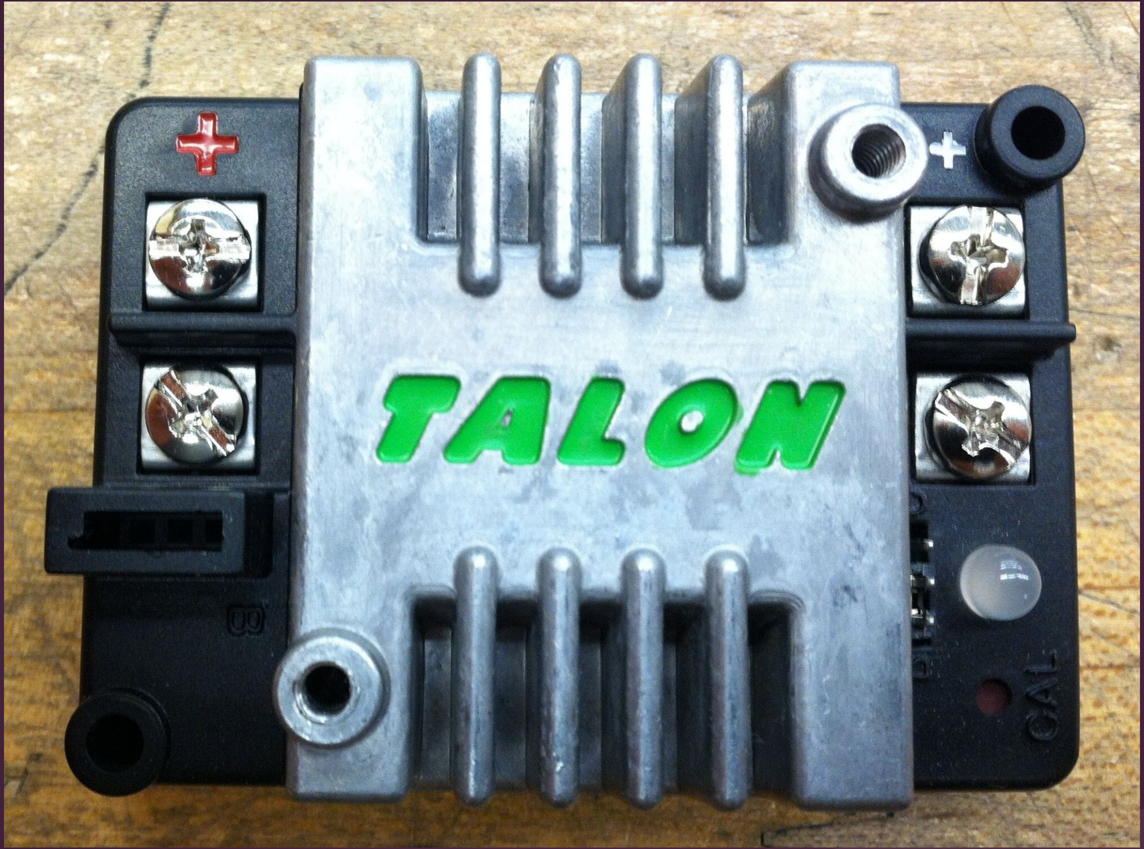


RoboRIO in Code

Parts of the RoboRIO in the API

- AnalogInput/AnalogOutput
- Preferences
- SerialPort





Talons

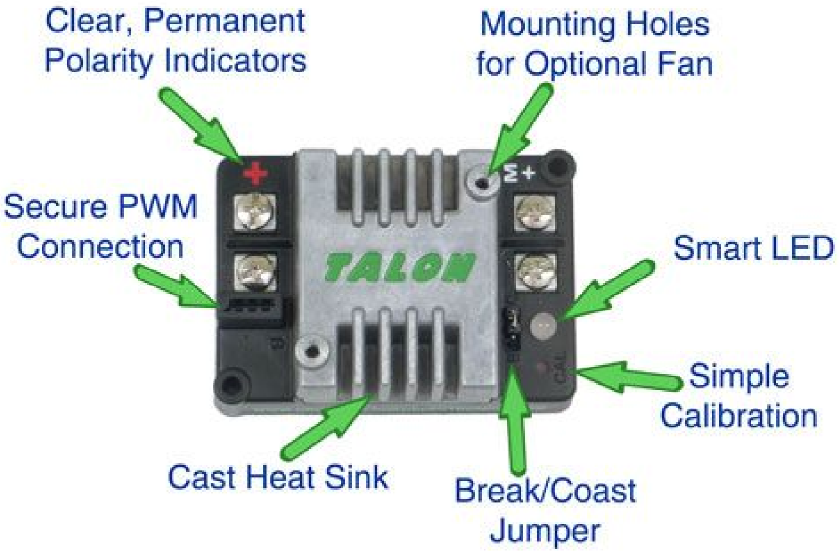
Talons in Code

- Talons have their own separate class in the 2015 API
- note that there's also CANTalon and TalonSRX--those are two other kinds of Talons!

How do you import Talons into code?

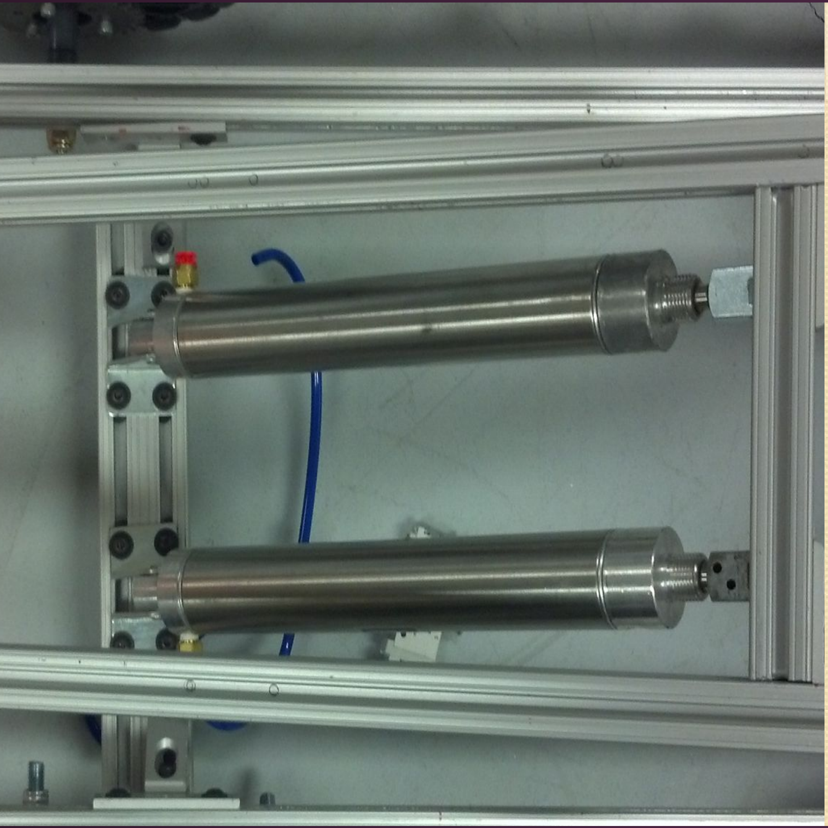
How do you construct a Talon object?

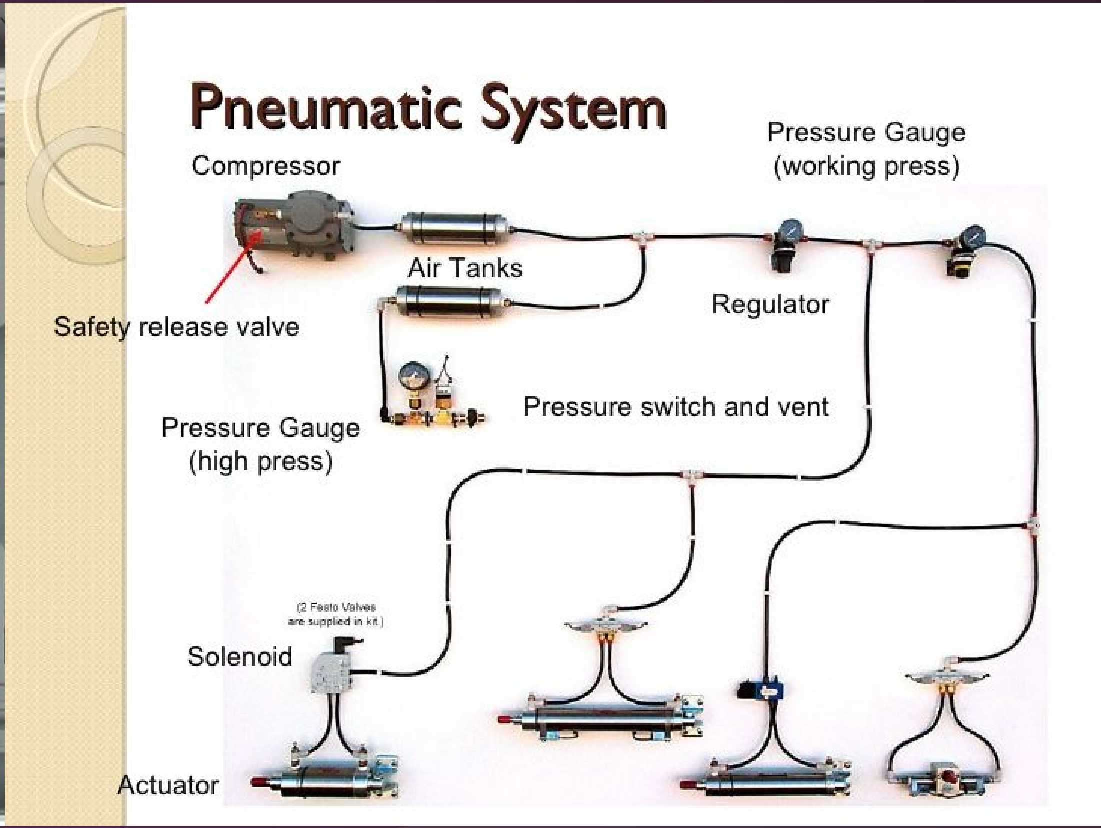
What methods can you use for your Talon object?



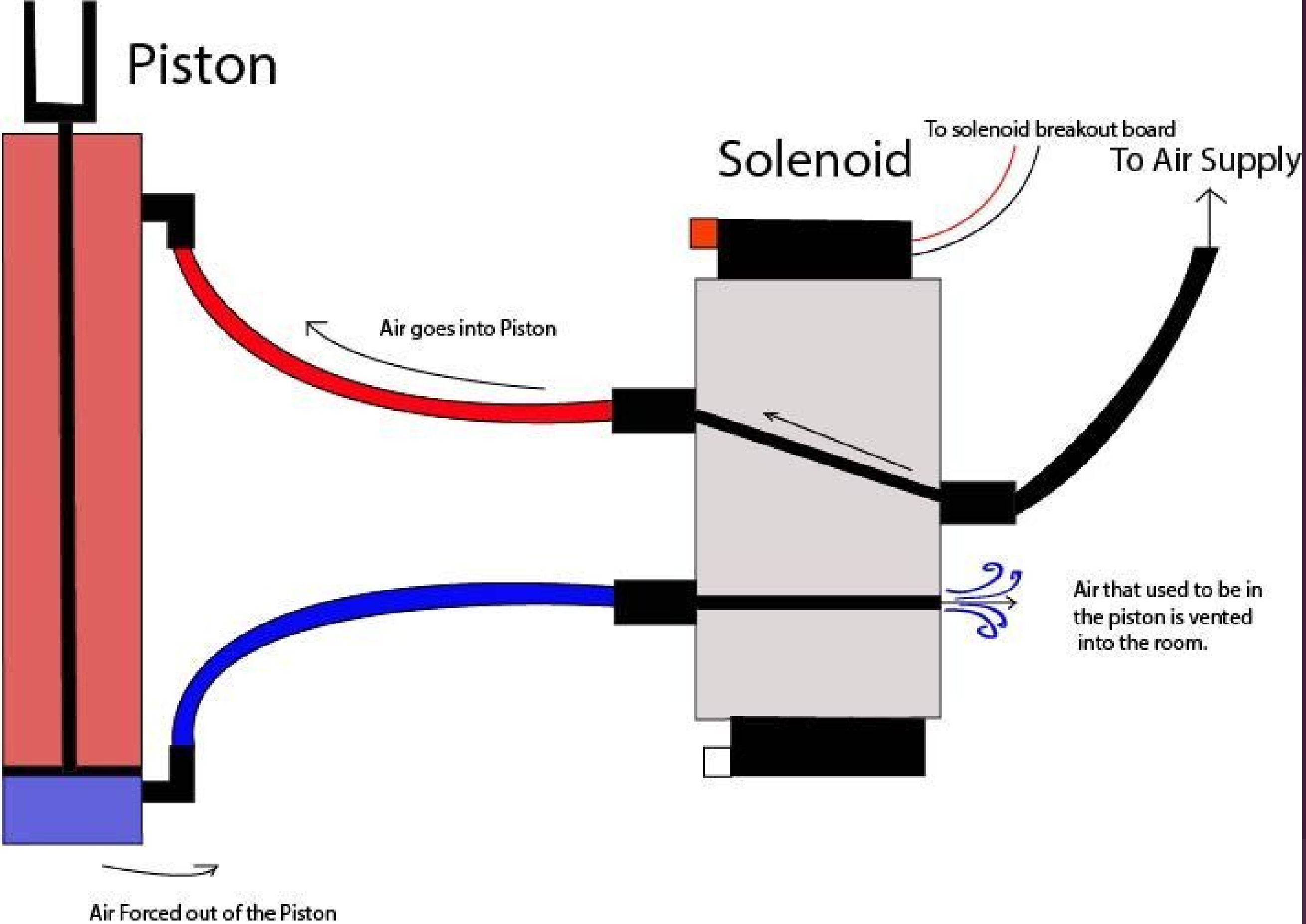
Talons in Code

- How do you import Talons into a code?
  - look at the top for the name of the library!
  - *import edu.wpi.first.wpilibj.Talon;*
- How do you construct a Talon object?
  - each class tells you how to construct an object
  - *Talon exampleTalon = new Talon(2)*;
- What methods can you use for your Talon object?
  - each class also lists all the methods you can use for the object
  - *get(), set(), pidWrite(), etc.*





Pistons + Solenoid


Piston
Solenoid
To solenoid breakout board
To Air Supply
Air goes into Piston
Air that used to be in the piston is vented into the room.
Air Forced out of the Piston

Pistons + Solenoid

Solenoids and Pistons in Code

- solenoids and pistons are two SEPARATE parts of the robot, but they work with one another
- this is really important to consider when writing code!!!!
- different parts of the robot can interact with one another in code and in real life

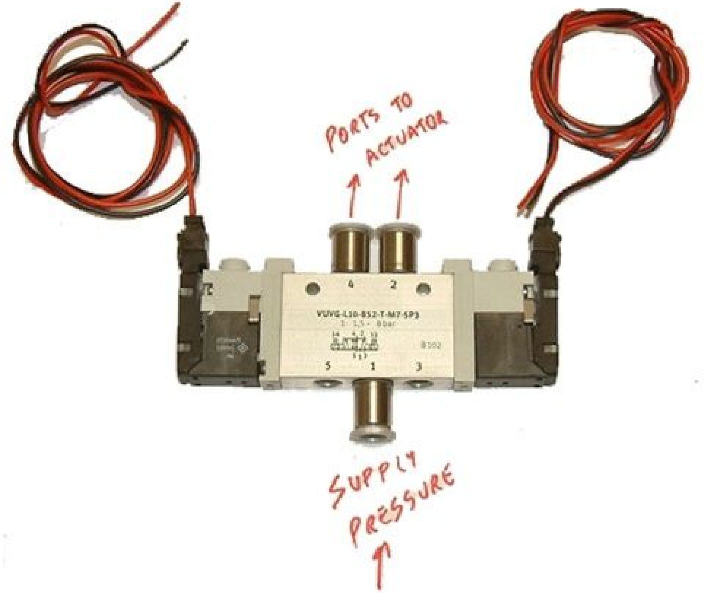
Solenoid Code

Let’s look at Solenoids first!

- Plain Solenoid VS. Double Solenoid
  - a plain solenoid connects to only one end of the piston, while a double solenoid can connect to both!
- if we used a plain solenoid, we have to create two solenoid objects to make a piston extend and retract.
- On the other hand, double solenoids only require one object to accomplish the same task!

Solenoid Code

- We have methods like get(), set(), isFwdSolenoidBlacklisted(), isRevSolenoidBlacklisted()
- What do the methods allow the solenoid to do?
- The methods allow us to get the states of the solenoid and manage values



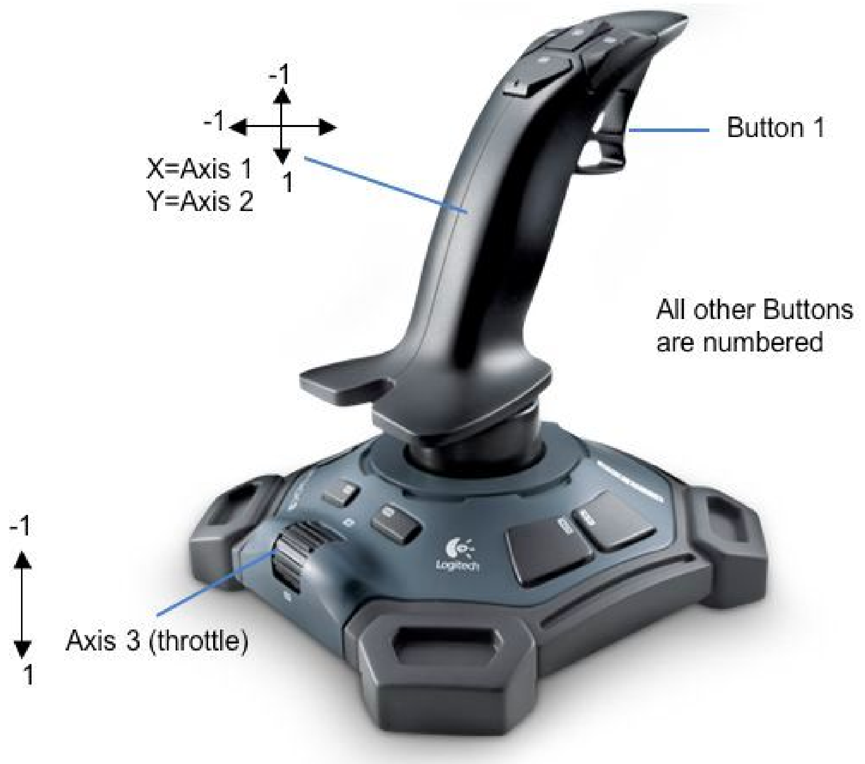
Solenoid to Compressor

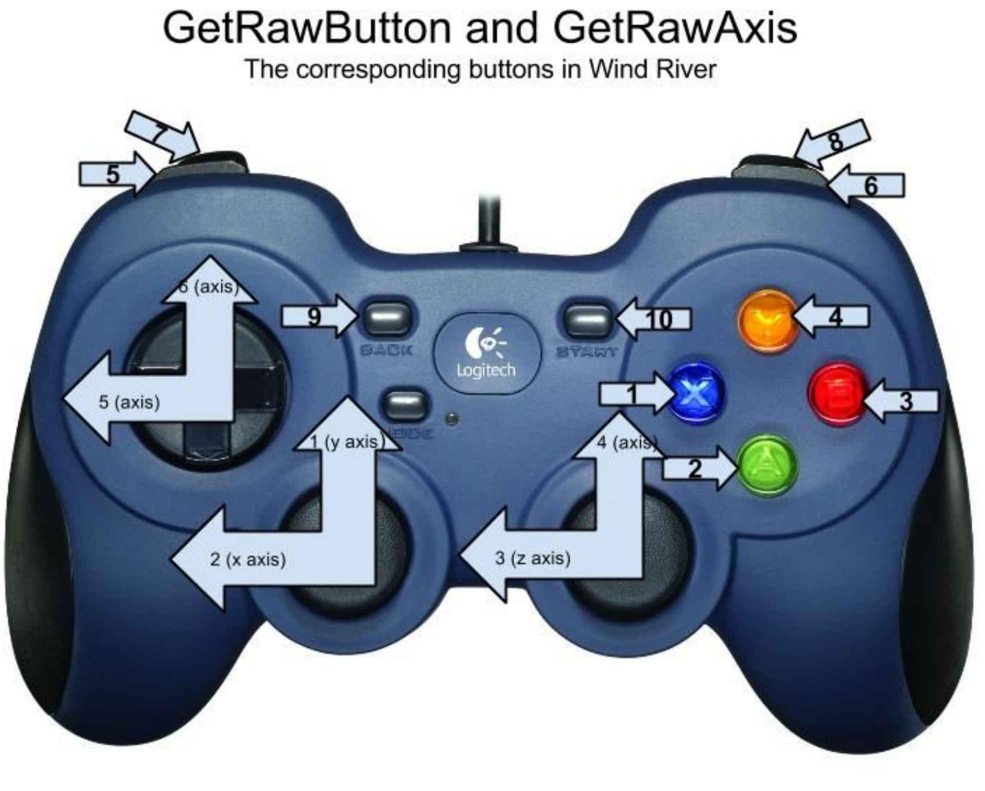
- the solenoid communicates with the compressor, which then collects and compresses air for the pistons
- we use the compressor class to communicate with the solenoid in the code

a compressor

Compressors in Code

- the compressor class helps us operate pneumatics
  - you’ll see the term Pneumatics Compressor Module (PCM) here, but if you noticed closely, PCM is frequently referred to in the Solenoid class!!!! (~wow~)
- compressor objects act in a loop
- PCM runs in a close-loop mode by default if you already created a Solenoid object
- note that a lot of the methods for compressors also keep track of the state of the compressor! (super important)




GetRawButton and GetRawAxis

The corresponding buttons in Wind River

Joysticks

Joysticks

- we use two types of joysticks, the one pictured on the right and xbox controllers
- these two joysticks are used for different actions
  - last year one joystick moved the robot while the other controlled the forklift
  - the captains drive the robot during competitions

our joystick looks like this except we have two of these

Joysticks in Code

- the joystick on the right moves 360 deg.
- it can return what direction the driver moves the stick with x and y coordinates
- it also tells you how far you moved it by returning values between 1 and -1
    - 1 is all the way forward, -1 is all the way backwards, 0 is resting
- look at the joystick methods to see what else you can do with it :)

our joystick looks like this except we have two of these

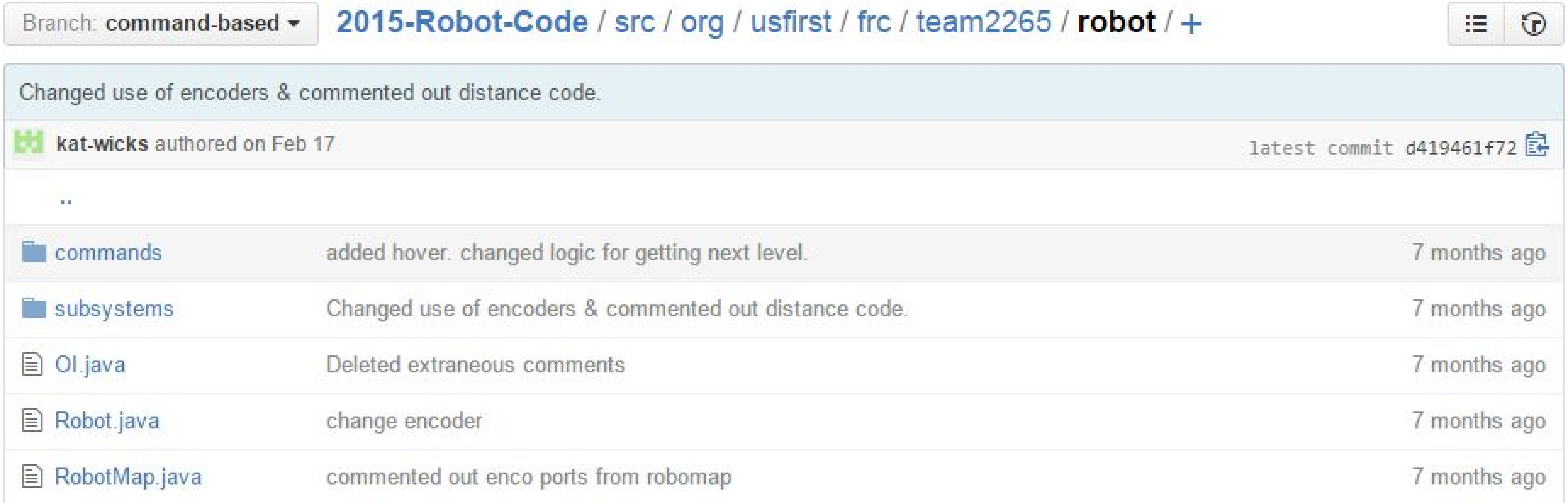
Writing Robot Code

Review: Subsystems/Commands

- when writing robot code, we need to decide which parts of the robot we need to code and if it’s a *subsystem* or a *command*
- *subsystems* are things each part of the robot can do
  - methods should define what sensors and actuators do
- *commands* are actions that use subsystems to accomplish a task

Putting Our Code Together

Our robot code on Github



Branch: **command-based** ▾ 2015-Robot-Code / src / org / usfirst / frc / team2265 / **robot** / +

Changed use of encoders & commented out distance code.

kat-wicks authored on Feb 17 latest commit d419461f72

..

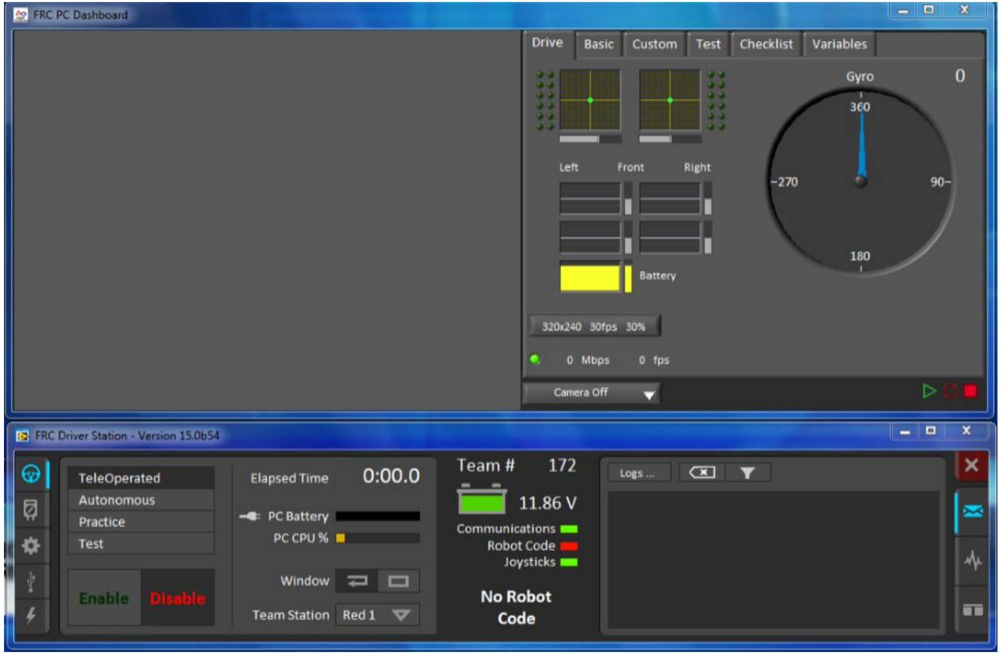
commands	added hover. changed logic for getting next level.	7 months ago
subsystems	Changed use of encoders & commented out distance code.	7 months ago
OI.java	Deleted extraneous comments	7 months ago
Robot.java	change encoder	7 months ago
RobotMap.java	commented out enco ports from robomap	7 months ago

Subsystems and Commands

- We use classes from the WPILib API to create our subsystems, each of which are classes
- We then use those subsystems to create our commands

Driver Station (DS)

- The driver station is a software from FIRST that allows us to run, test, and debug our codes



FRC PC Dashboard

Gyro 0

320x240 30fps 30%

0 Mbps 0 fps

Camera Off

FRC Driver Station - Version 15.0b54

Elapsed Time 0:00.0

Team # 172

11.86 V

Window

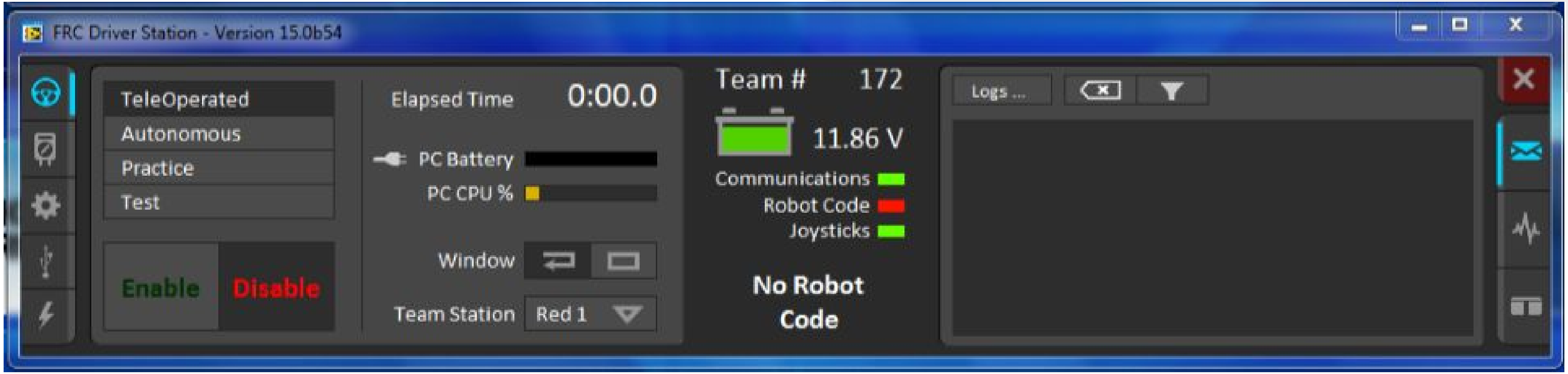
Team Station Red 1

No Robot Code

dashboard

driver station

Driver Station Features


FRC Driver Station - Version 15.0b54

Elapsed Time 0:00.0

Window

Team Station Red 1

Team # 172

11.86 V

Communications 🟩

Robot Code 🟥

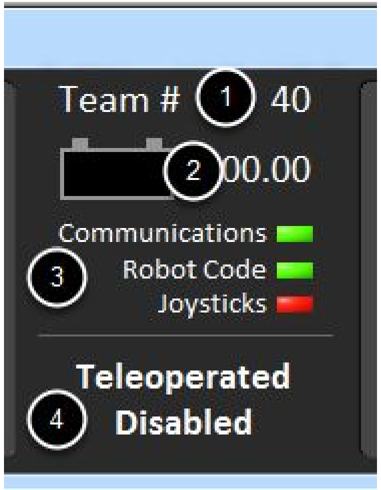
Joysticks 🟩

No Robot Code

Status pane, operation tab, diagnostic tab, setup tab, USB devices tab, CAN/power tab

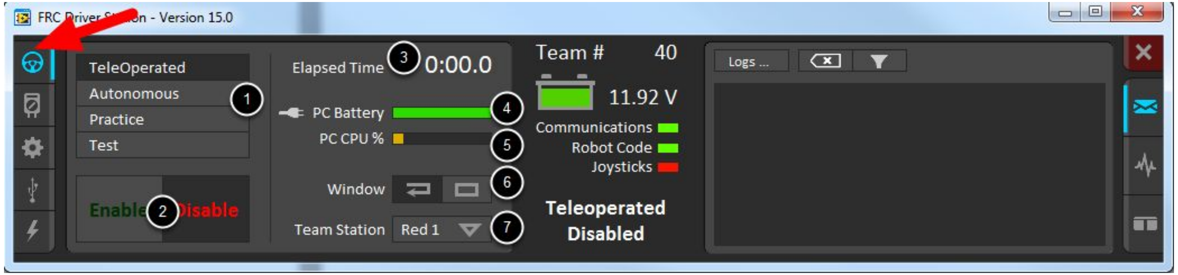
Status Pane

- Displays critical information of the DS and robot
- **Team number**: should display 2265
- **Battery voltage**: display battery voltage as #
- **Major status indicator**: shows status of three things - if DS is communicating with RoboRIO, if robot code is running, and if joystick(s) is plugged in & recognized
- **Status string**: indicates status of the robot overall



Driver Station Features

Operation tab: lets you control mode of robot and provide additional status indicators



FRC Driver Station - Version 15.0

Elapsed Time 0:00.0

PC Battery

PC CPU %

Team Station Red 1

Team # 40

11.92 V

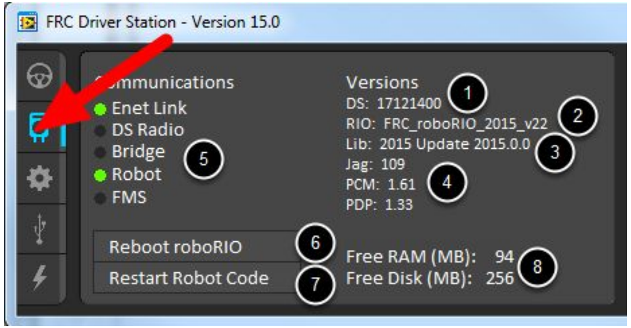
Communications 🟩

Robot Code 🟩

Joysticks 🟥

Teleoperated Disabled

Diagnostics tab: provides status indicators to help run diagnostic issues with robot



FRC Driver Station - Version 15.0

Communications

● Enet Link

● DS Radio

● Bridge

● Robot

● FMS

Versions

DS: 17121400

RIO: FRC_roboRIO_2015_v22

Lib: 2015 Update 2015.0.0

Jag: 109

PCM: 1.61

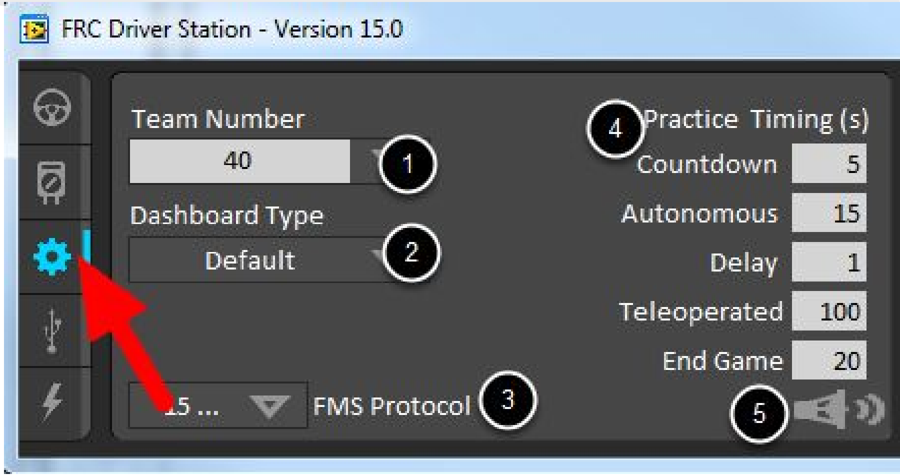
PDP: 1.33

Free RAM (MB): 94

Free Disk (MB): 256

Driver Station Features

Setup tab: can control several operations on the DS



FRC Driver Station - Version 15.0

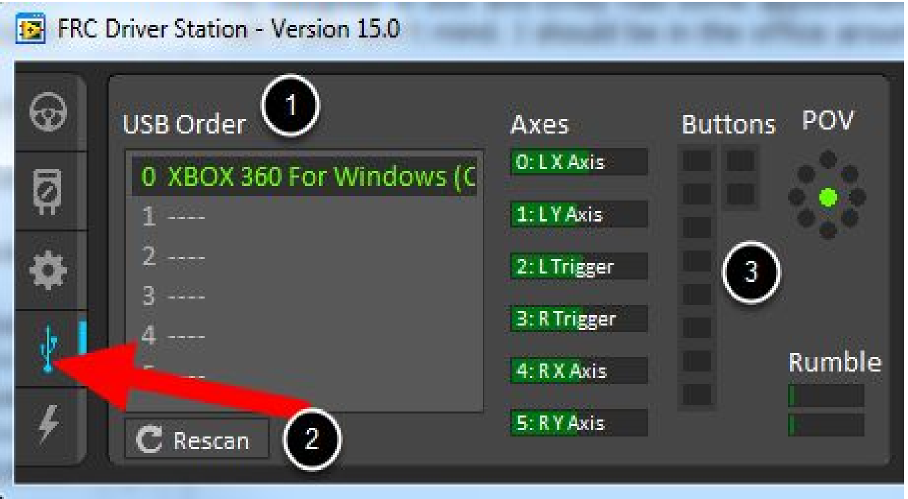
Team Number: 40

Dashboard Type: Default

Practice Timing (s)

Countdown	5
Autonomous	15
Delay	1
Teleoperated	100
End Game	20

15 ... FMS Protocol



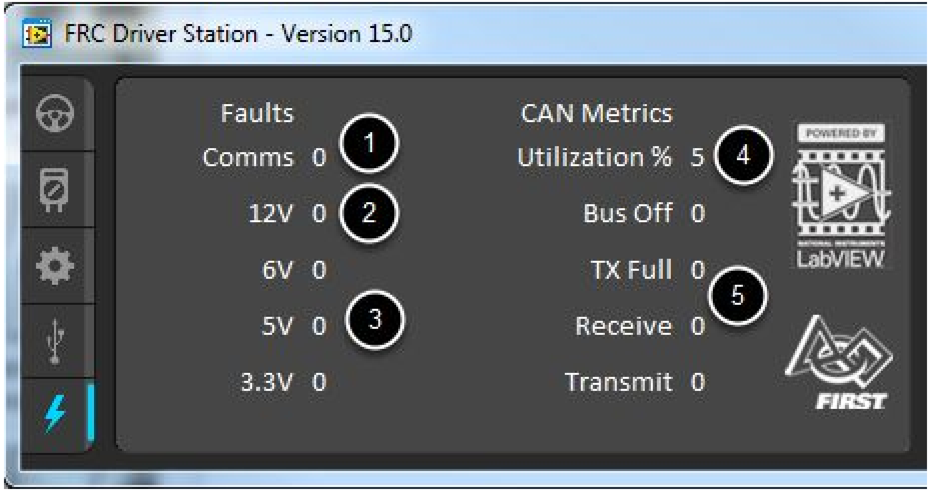
FRC Driver Station - Version 15.0

USB Order
0 XBOX 360 For Windows (C
1 ----
2 ----
3 ----
4 ----

Axes
0: L X Axis
1: L Y Axis
2: L Trigger
3: R Trigger
4: R X Axis
5: R Y Axis

USB Device Tab: Lists information of all USB devices connected to DS

CAN/Power Tab



FRC Driver Station - Version 15.0

Faults

Comms 0

12V 0

6V 0

5V 0

3.3V 0

CAN Metrics

Utilization % 5

Bus Off 0

TX Full 0

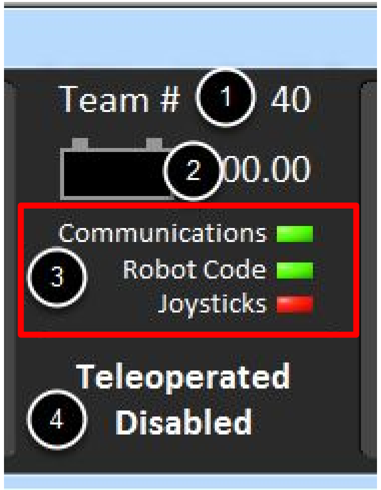
Receive 0

Transmit 0

- Shows power status of RoboRIO and status of CAN box
- Has additional tabs on the side to show various graphs to help us diagnose robot issues

Running Robot Code

- Open DS and see the major status indicator
- All three will usually be red at first
- We have to get all three parts green to start working



Team # 40

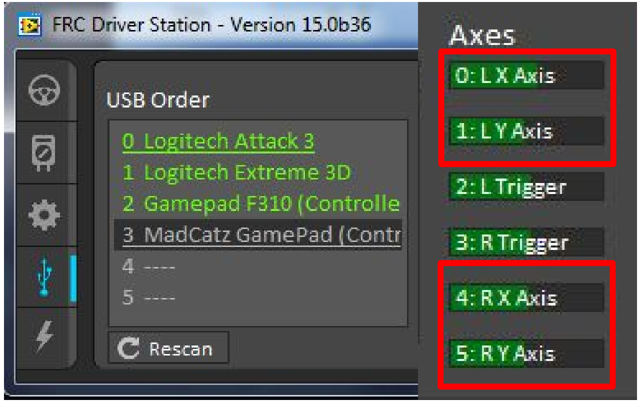
00.00

Communications 🟩
Robot Code 🟩
Joysticks 🟥

Teleoperated
Disabled

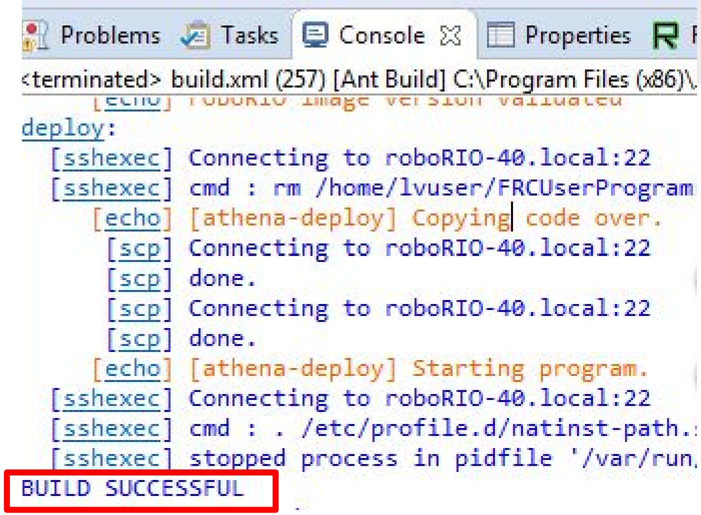
Running Robot Code (Joysticks)

- all we need to do is plug the joysticks in and the major status indicator will turn green
- we need to go to see if the X and Y Axes of the Joysticks are at resting position on the DS (as seen on the right)
- if not, just unplug and replug



Running Robot Code (Eclipse)

- now that the joysticks are green, we’ll get the robot code to run!
- open eclipse and go to the Robot.java file
- Click on Run → Run As → WPIlib Deploy
- If it says build successful, then the major status indicator should be green
- if not, see where the errors are
- if it’s not a code error you can troubleshoot by googling the error you get



```
<terminated> build.xml (257) [Ant Build] C:\Program Files (x86)\
     [echo] roboRIO image version validated
deploy:
  [sshexec] Connecting to roboRIO-40.local:22
  [sshexec] cmd : rm /home/lvuser/FRCUserProgram
     [echo] [athena-deploy] Copying code over.
      [scp] Connecting to roboRIO-40.local:22
      [scp] done.
      [scp] Connecting to roboRIO-40.local:22
      [scp] done.
     [echo] [athena-deploy] Starting program.
  [sshexec] Connecting to roboRIO-40.local:22
  [sshexec] cmd : . /etc/profile.d/natinst-path.
  [sshexec] stopped process in pidfile '/var/run,
BUILD SUCCESSFUL
```

Robot Code (Communications)

- now we only have communications left!
- we have to connect the radio to RoboRIO, so we can just log in as mentioned before
- from there we can test!

Homework

Use the 2015 API to create code for a robot that can pick up a soda can, move, and then place it somewhere else.

Remember to consider:

- what parts of the robot you need to create
- what classes from the API do you need to make those parts?
- what are the subsystems and the commands?
- what are the methods for each subsystem?

Homework

Don’t forget to import things like commands, subsystems, SmartDashboard

Example Answers

Subsystems: Claw, motors, joysticks

Commands: PlaceSoda, PrepareToGrab, Grab, DriveWithJoysticks